Chapter 3 Convolutional Codes and Trellis Coded Modulation

- 3.1 Encoder Structure and Trellis Representation
- 3.2 Systematic Convolutional Codes
- 3.3 Viterbi Decoding Algorithm
- 3.4 BCJR Decoding Algorithm
- 3.5 Trellis Coded Modulation

§ 3.1 Encoder Structure and Trellis Representation

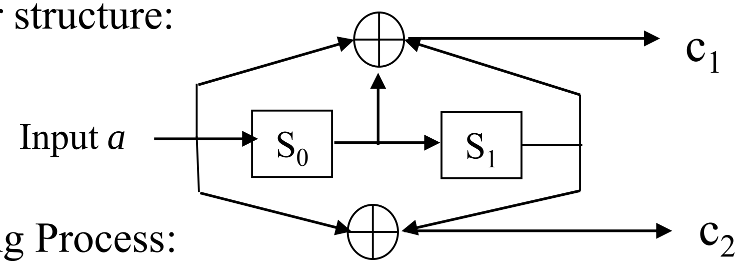
- Introduction
  - Encoder: contains memory (order *m*: *m* memory units);
  - Output: encoder output at time unit *t* depends on the input and the memory units status at time unit *t*;
  - By increasing the memory order *m*, one can increase the convolutional code's minimum distance (d_{min}) and achieve low bit error rate performance (P_b);
  - Decoding Methods:
    - Viterbi algorithm [1]: Maximum Likelihood (ML) decoding algorithm;
    - Bahl, Cocke, Jelinek, and Raviv (BCJR) [2] algorithm: Maximum *A Posteriori* Probability (MAP) decoding algorithm, used for iterative decoding process, e.g. turbo decoding.

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Trans. Inform. Theory, IT-13, 260-269, April, 1967.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans, Inform. Theory, IT-20; 284-287, March, 1974.

§ 3.1 Encoder Structure and Trellis Representation

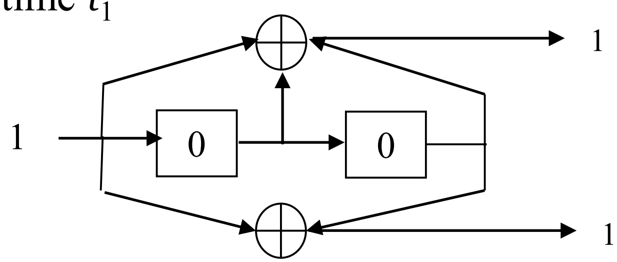
- The $(7, 5)_8$ conv. code
  - Encoder structure:



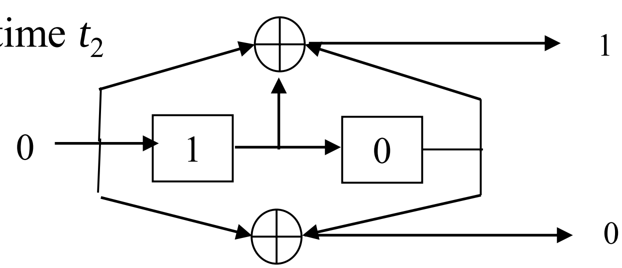
  - Encoding Process:

(Initialised state $s_0s_1 = 00$)

At time t_1



At time t_2



Code rate: ½;

Memory: $m = 2$;

Constraint length: $m + 1 = 3$

Output calculation:

$c_1 = a \oplus S_0 \oplus S_1$;

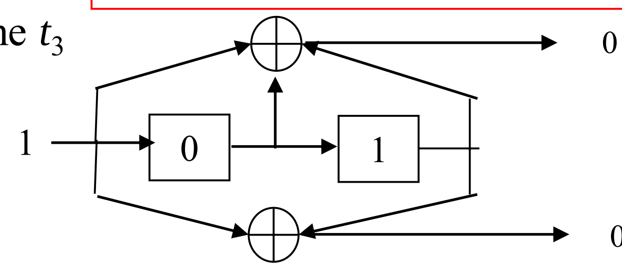
$c_2 = a \oplus S_1$;

Registers update:

$S_1' = S_0$.

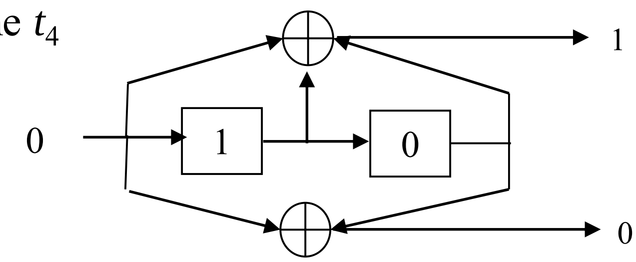
$S_0' = a$.

At time t_3

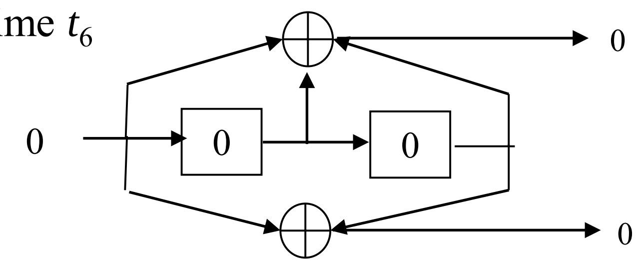


§ 3.1 Encoder Structure and Trellis Representation

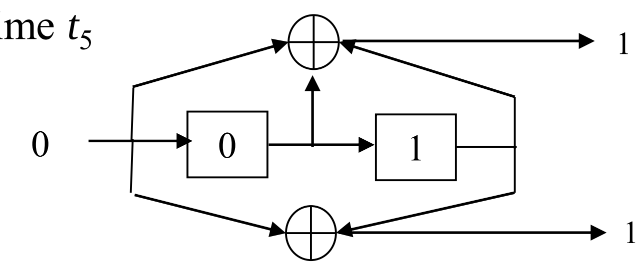
At time t_4



At time t_6



At time t_5

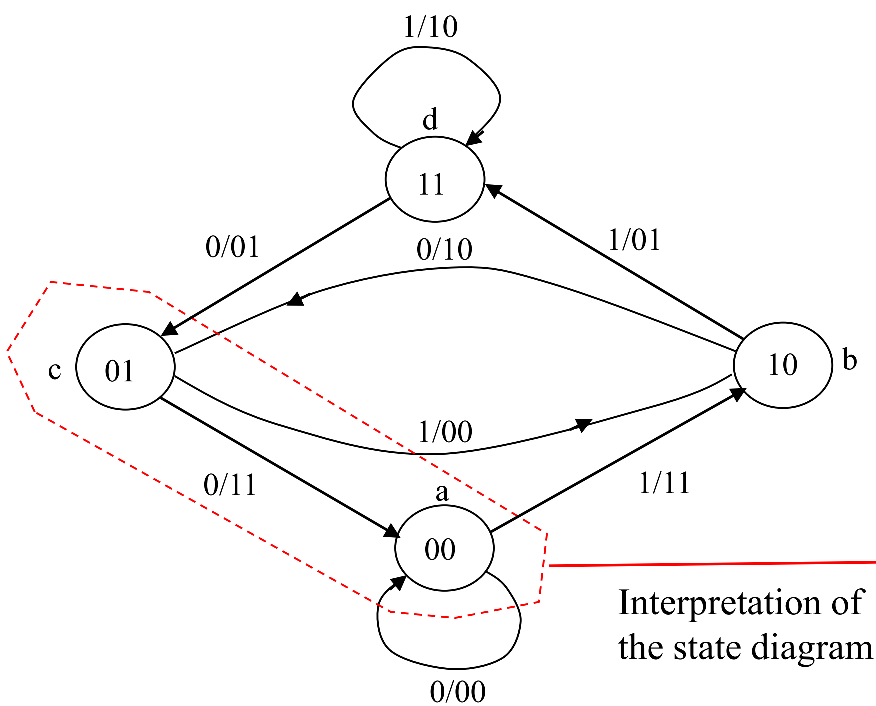


Input sequence $[m_1\ m_2\ m_3\ m_4\ m_5\ m_6] = [1\ 0\ 1\ 0\ 0\ 0]$

Output sequence $[c_1^1 c_1^2\ \ c_2^1 c_2^2\ \ c_3^1 c_3^2\ \ c_4^1 c_4^2\ \ c_5^1 c_5^2\ \ c_6^1 c_6^2] = [11\ 10\ 00\ 10\ 11\ 00]$

§ 3.1 Encoder Structure and Trellis Representation

A state transition diagram of the $(7, 5)_8$ conv. code



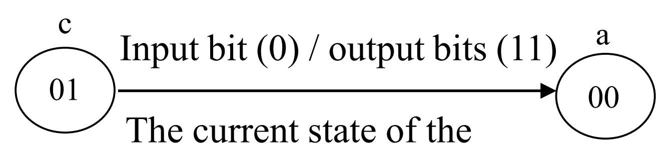
State definition (s_0 s_1)

a = 00

b = 10

c = 01

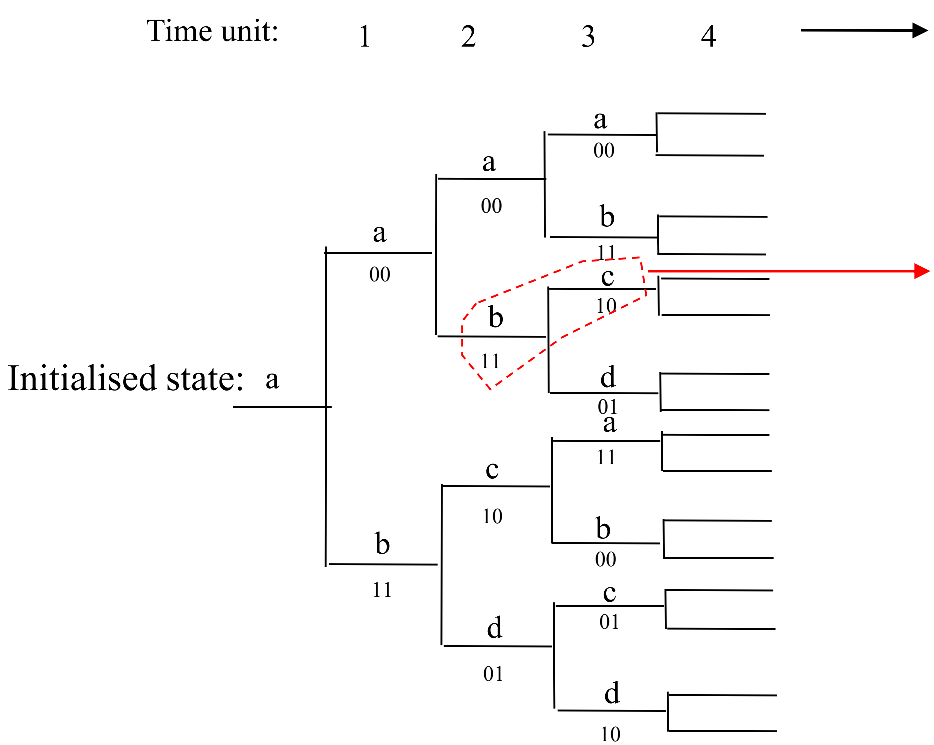
d = 11



The current state of the encoder is **<u>c</u>**. If the input bit is **<u>0</u>**, it will output **<u>11</u>** and the next state of the encoder is **<u>a</u>**.

§ 3.1 Encoder Structure and Trellis Representation

Tree Representation of the $(7, 5)_8$ conv. code



Tree diagram interpretation:

The current state of the encoder is **<u>b</u>**. If the input bit is **<u>0</u>**, the output will be **<u>10</u>**, and the next state of the encoder is **<u>c</u>**.

↑ Input bit as 0

↓ Input bit as 1

$\frac{\text{State after transition}}{\text{Output from transition}}$

Example 3.1 Determine the codeword that corresponds to message [0 1 1 0 1]

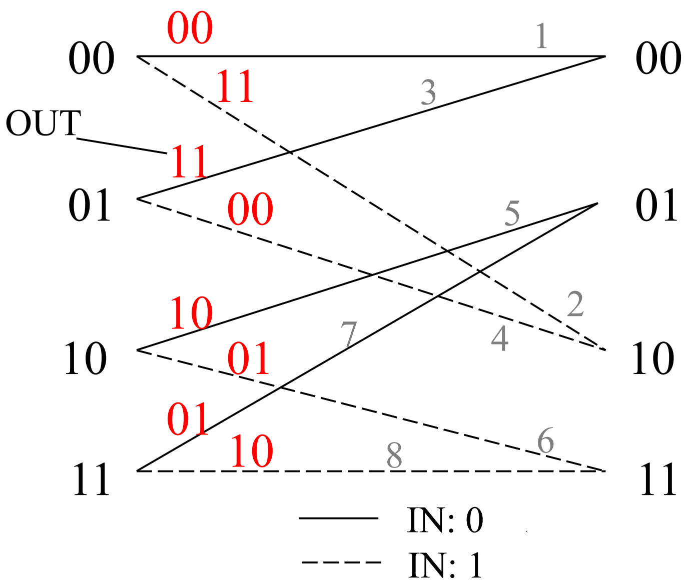
§ 3.1 Encoder Structure and Trellis Representation

Trellis of the $(7, 5)_8$ conv. code

State Table

IN	Current State	Next State	Out	ID
0	00	00	00	1
1	00	10	11	2
0	01	00	11	3
1	01	10	00	4
0	10	01	10	5
1	10	11	01	6
0	11	01	01	7
1	11	11	10	8

Trellis



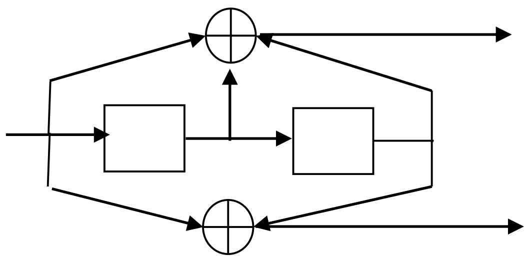
Remark: A trellis tells the state transition and IN/OUT relationship. It can be used to yield a convolutional codeword of a sequential input.

Example 3.2 Use the above trellis to determine the codeword that corresponds to message [0 1 1 0 1].

§ 3.1 Encoder Structure and Trellis Representation

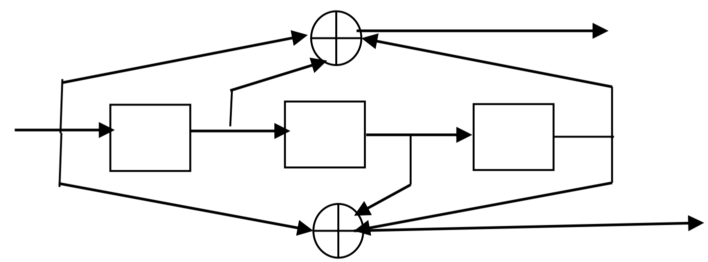
A number of conv. codes

$(7, 5)_8$ conv. code



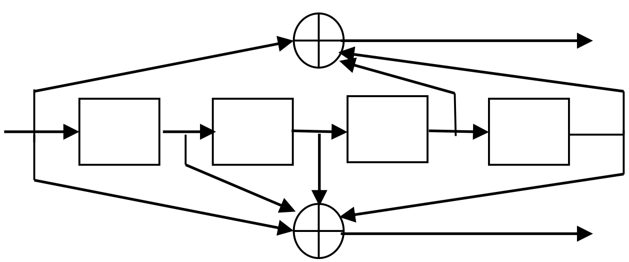
4 states

$(15, 13)_8$ conv. code



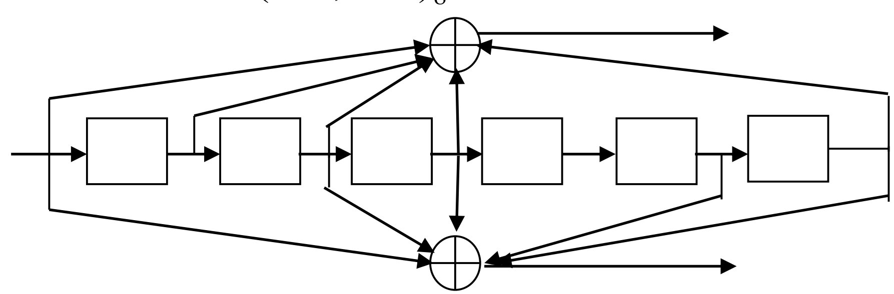
8 states

$(23, 35)_8$ conv. code



16 states

$(171, 133)_8$ conv. code



64 states

Remark: A convolutional code's error-correction capability improves by increasing the number of the encoder states.

§ 3.1 Encoder Structure and Trellis Representation

Remark: The encoder structure can also be represented by generator sequences or transfer functions.

Example 3.3: The $(7, 5)_8$ conv. code can also be written as:

A rate $^1/_2$ conv. code with generator sequences

$$g^{(0)} = [1\ 1\ 1], \qquad g^{(1)} = [1\ 0\ 1].$$

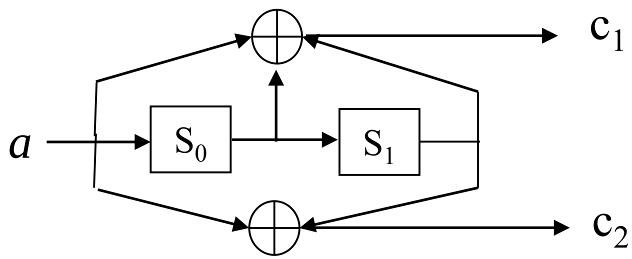
A rate $^1/_2$ conv. code with transfer functions:

$$g^{(1)}(x) = 1 + x + x^2, \quad g^{(2)}(x) = 1 + x^2.$$

§ 3.2 Systematic Convolutional Codes

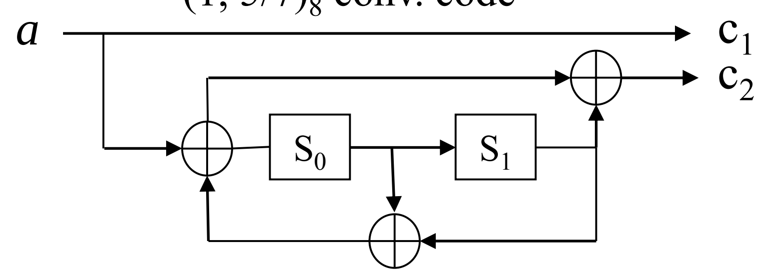
- The $(7, 5)_8$ conv. code's systematic counterpart is:

$(7, 5)_8$ conv. code



Nonsystematic code

$(1, 5/7)_8$ conv. code



Systematic code

Encoding and Registers' updating rules:

$[S_0\ S_1]$ are initialization as [0 0];

$c_1 = a$; (systematic feature) feedback $= S_0 \oplus S_1$;

$c_2 = a \oplus$ feedback $\oplus S_1$; $S_1' = S_0$; $S_0' = a \oplus$ feedback;

Remark: Systematic encoding structure is important for iterative decoding, e.g., the decoding of turbo codes.

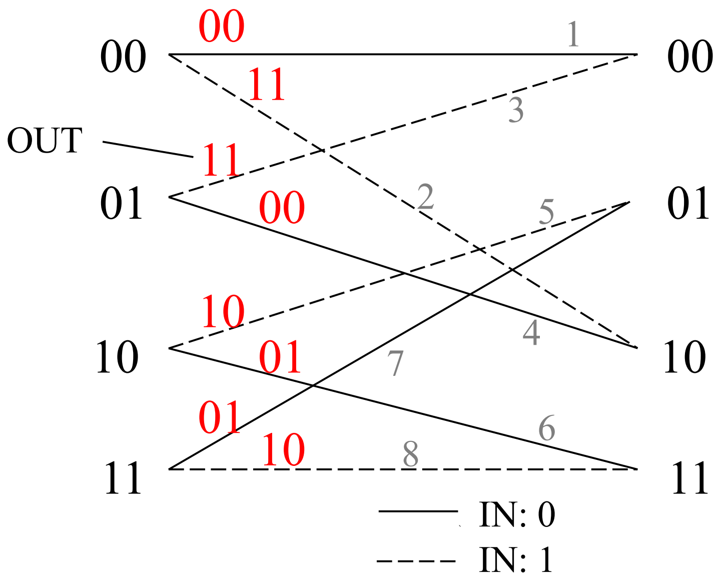
§ 3.2 Systematic Convolutional Codes

For the $(1, 5/7)_8$ conv. code

State Table

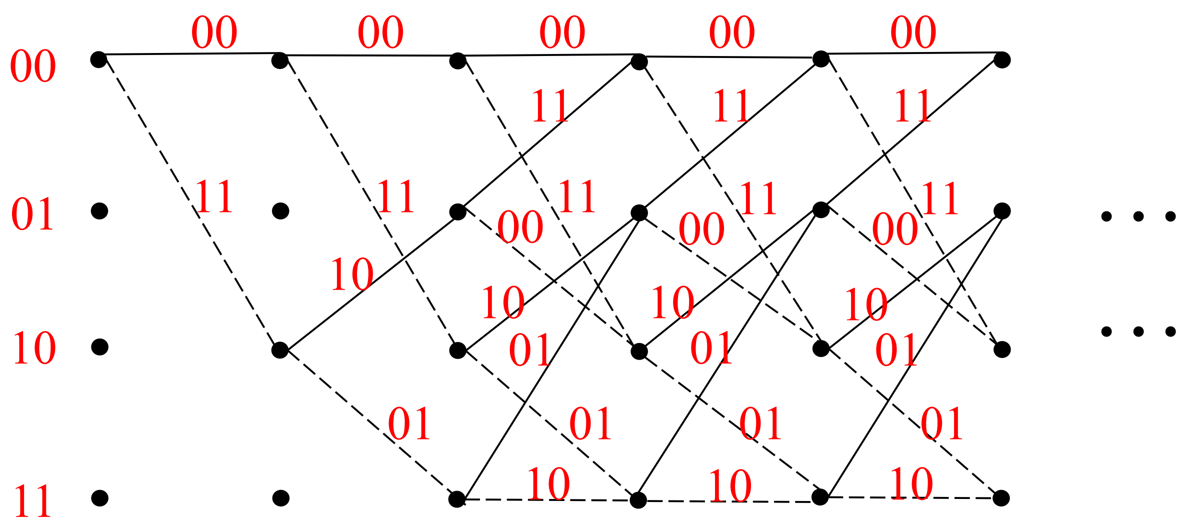
IN	Current State	Next State	Out	ID
0	00	00	00	1
1	00	10	11	2
0	01	10	00	4
1	01	00	11	3
0	10	11	01	6
1	10	01	10	5
0	11	01	01	7
1	11	11	10	8

Trellis



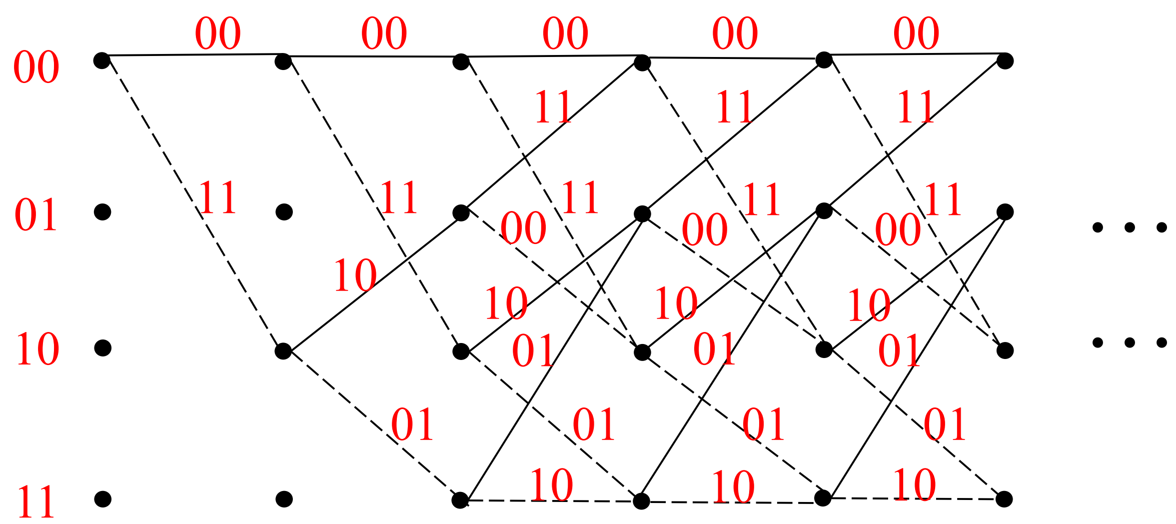
§ 3.3 Viterbi Decoding Algorithm

Let us extend the trellis of the $(7, 5)_8$ conv. code as if there is a sequential input.



- Such an extension results in a **Viterbi trellis**
- A path in the Viterbi trellis represents a convolutional codeword that corresponds to a sequential input (message).

§ 3.3 Viterbi Decoding Algorithm



- Decoding motivation: Given a received word $\bar{R}$, find the mostly likely codeword $\hat{\bar{C}}$ such that the Hamming distance $d_{Ham}(\bar{R}, \hat{\bar{C}})$ is minimized.
- Since $\hat{\bar{C}}$ corresponds to a path in the Viterbi trellis, trace back the path of $\hat{\bar{C}}$ enable us to find out the message.
- Branch metrics: Hamming distance between a transition branch's output and the corresponding received symbol (or bits).
- Path metrics: Accumulated Hamming distance of the previous branch metrics.

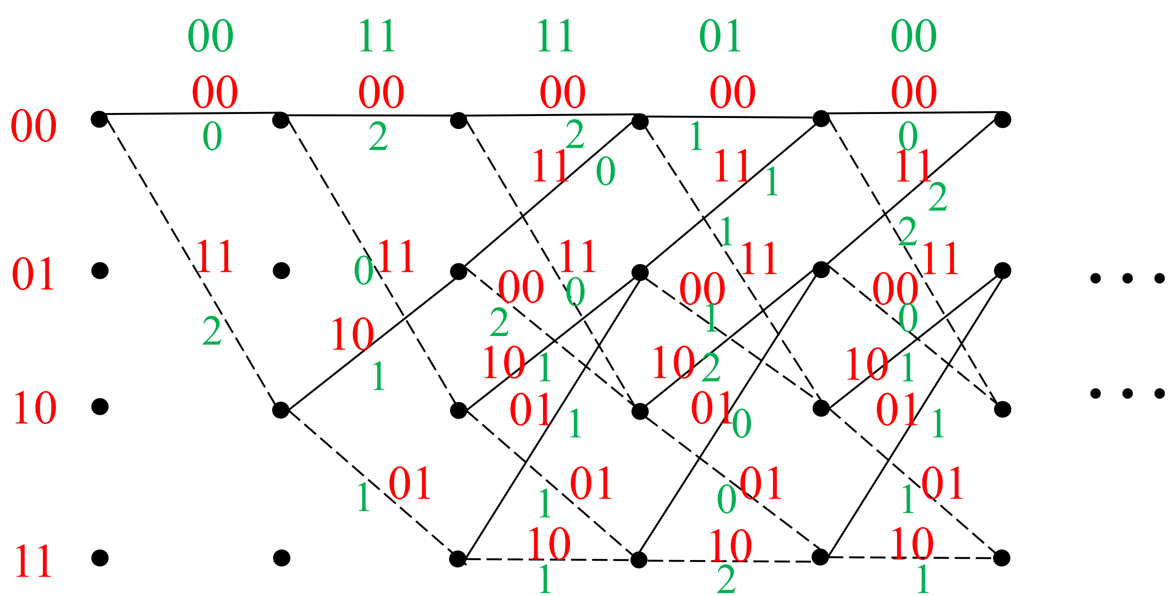
§ 3.3 Viterbi Decoding Algorithm

Example 3.4. Given the $(7, 5)_8$ conv. code as in Examples 3.1-3.3. The transmitted codeword is

$\bar{C}$ =[0 0 1 1 01 01 00]. After channel, the received word is

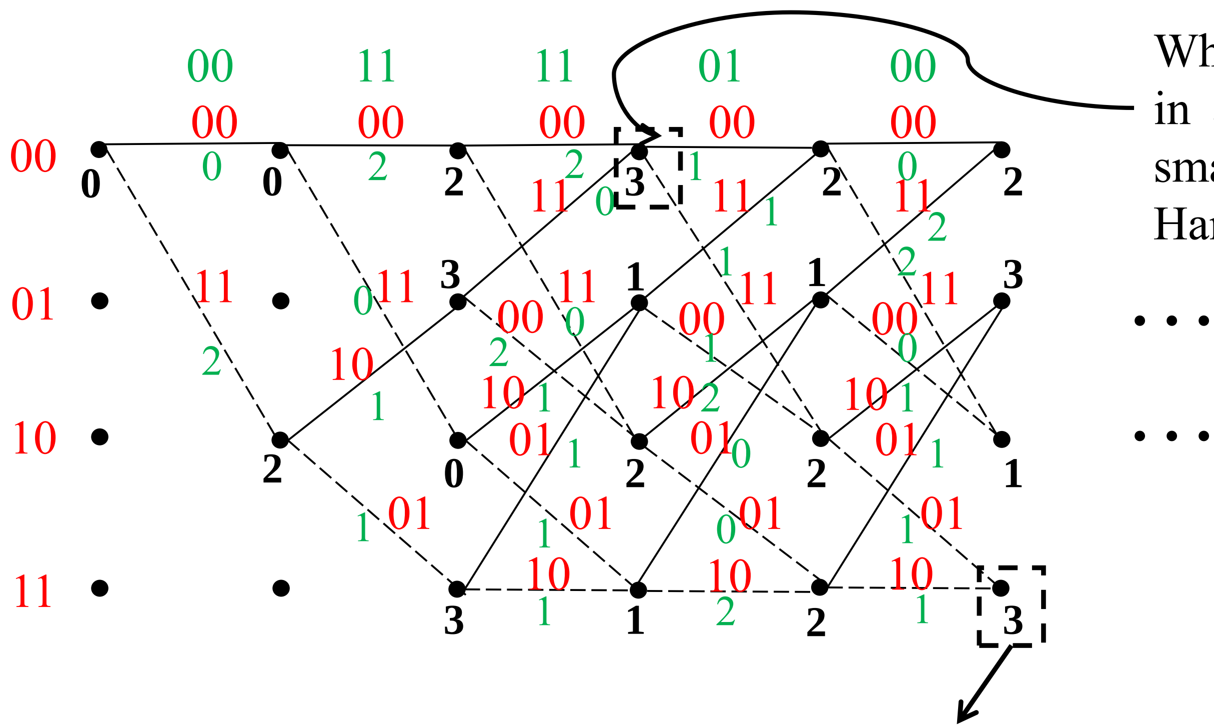
$\bar{R}$ =[0 0 1 1 $\boxed{1}$1 01 00]. Try to use the Viterbi trellis to decode it.

Step 1: Calculate all the branch metrics.



§ 3.3 Viterbi Decoding Algorithm

Step 2: Calculate the path metrics.



When two paths join in a node, keep the smaller accumulated Hamming distance.

When the two joining paths give the same accumulated Hamming distance, pick up one randomly.

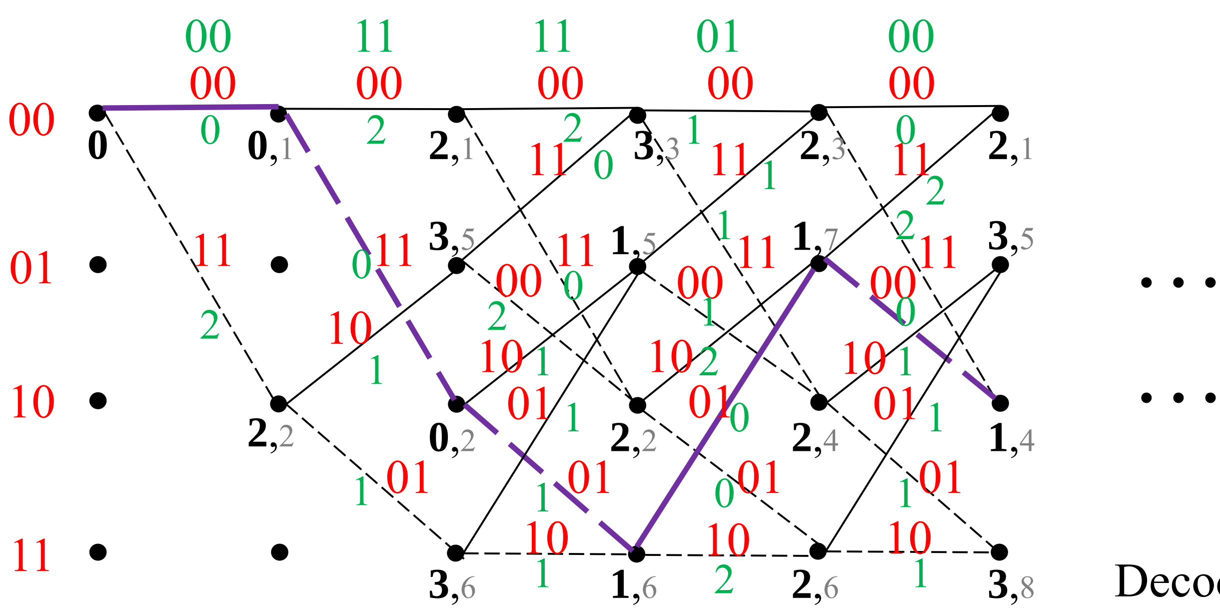
§ 3.3 Viterbi Decoding Algorithm

Step 3: Pick up the minimal path metric and trace back to determine the message.

Tracing rules: (1) Trellis connection;

(2) The previous path metric should NOT be greater than the current path metric;

(3) The tracing route should match the trellis transition ID.



Decoding output: 0 1 1 0 1

§ 3.3 Viterbi Decoding Algorithm

Branch Metrics Table

0	2	2	1	0
∞	∞	0	1	2
2	0	0	1	2
∞	∞	2	1	0
∞	1	1	2	1
∞	∞	1	0	1
∞	1	1	0	1
∞	∞	1	2	1

Path Metrics Table

0	0	2	3	2	2
∞	∞	3	1	1	3
∞	2	0	2	2	1
∞	∞	3	1	2	3

Trellis Transition ID Table

1	1	3	3	1
×	5	5	7	5
2	2	2	4	4
×	6	6	6	8

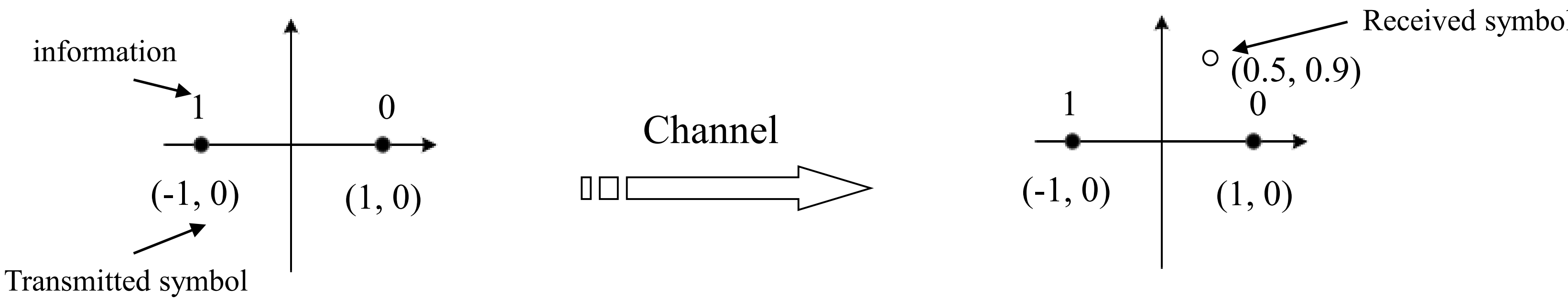
§ 3.3 Viterbi Decoding Algorithm

Soft-decision Viterbi decoding

- While we are performing the hard-decision Viterbi decoding, we have the scenario that two joining paths yield the same accumulated Hamming distance. This would cause decoding ‘ambiguity’ and performance penalty;
- Such a performance loss can be compensated by utilizing soft-decision decoding, e.g., soft-decision Viterbi decoding

Modulation and Demodulation (e.g., BPSK)

- Modulation: mapping binary information into a transmitted symbol;
- Demodulation: determining the binary information with a received symbol;

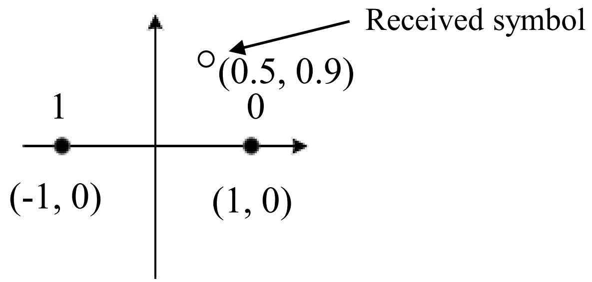


Modulation

Demodulation

§ 3.3 Viterbi Decoding Algorithm

➢ Modulation and Demodulation (e.g., BPSK)



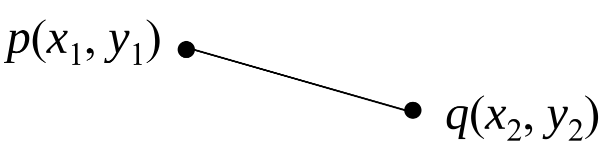
Demodulation

Hard-decision: the information bit is 0. The Hamming distance becomes the Viterbi decoding metrics;

Soft-decision: the information bit has Pr. of 0.7 being 0 and Pr. of 0.3 bing 1. The *Euclidean distance* (*or probability*) becomes the Viterbi decoding metrics;

➢ Euclidean Distance

Definition: The Euclidean distance between points *p* and *q* is the length of the line segment connecting them.



$$d_{Eud} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

§ 3.3 Viterbi Decoding Algorithm

Example 3.5. Given the $(7, 5)_8$ conv. code as in Examples 3.1-3.3. The transmitted codeword is $\bar{C} = [0\ 0\ \ 1\ 1\ \ 0\ 1\ \ 0\ 1\ \ 00]$.

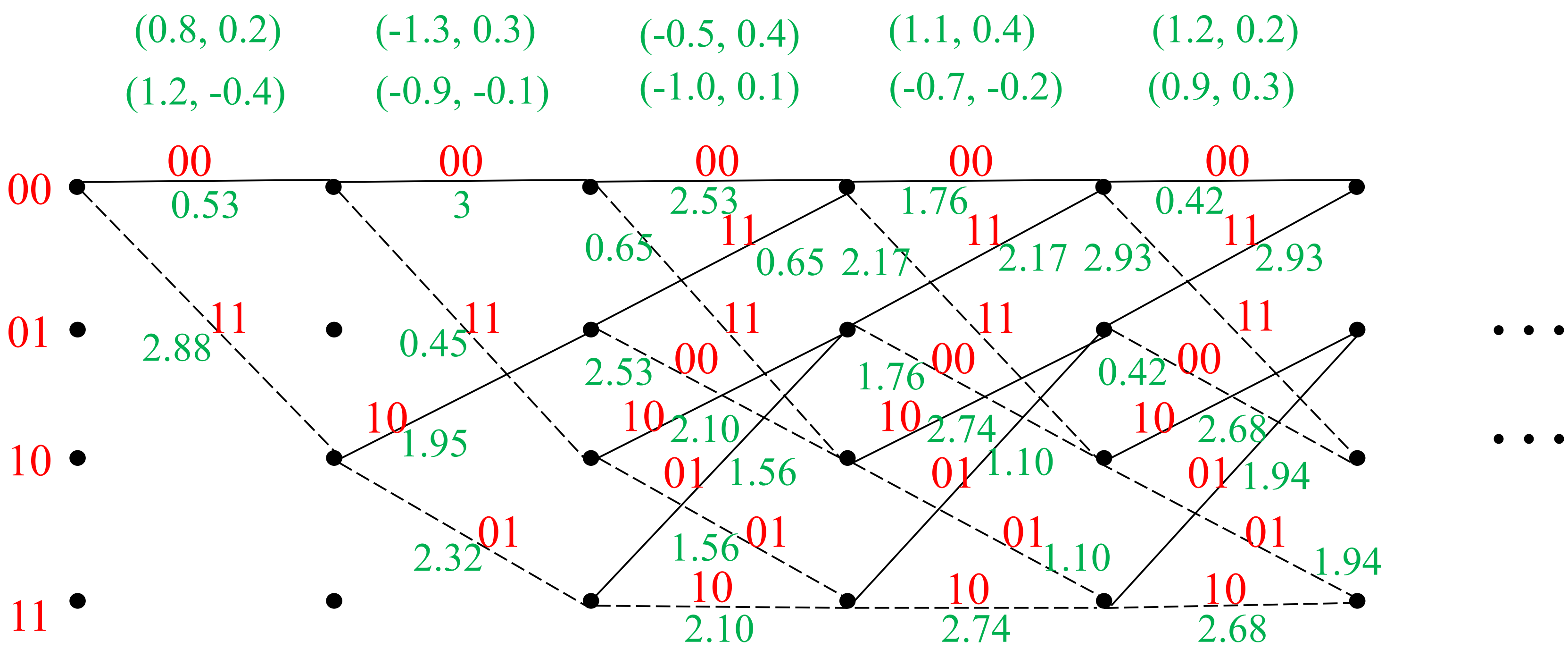
After BPSK modulation, the transmitted symbols are:
(1, 0), (1, 0), (-1, 0), (-1, 0), (1, 0), (-1, 0), (1, 0), (-1, 0), (1, 0), (1, 0).

After the channel, the received symbols are:
(0.8, 0.2), (1.2, -0.4), (-1.3, 0.3), (-0.9, -0.1), (-0.5, 0.4), (-1.0, 0.1), (1.1, 0.4), (-0.7, -0.2), (1.2, 0.2), (0.9, 0.3).

§ 3.3 Viterbi Decoding Algorithm

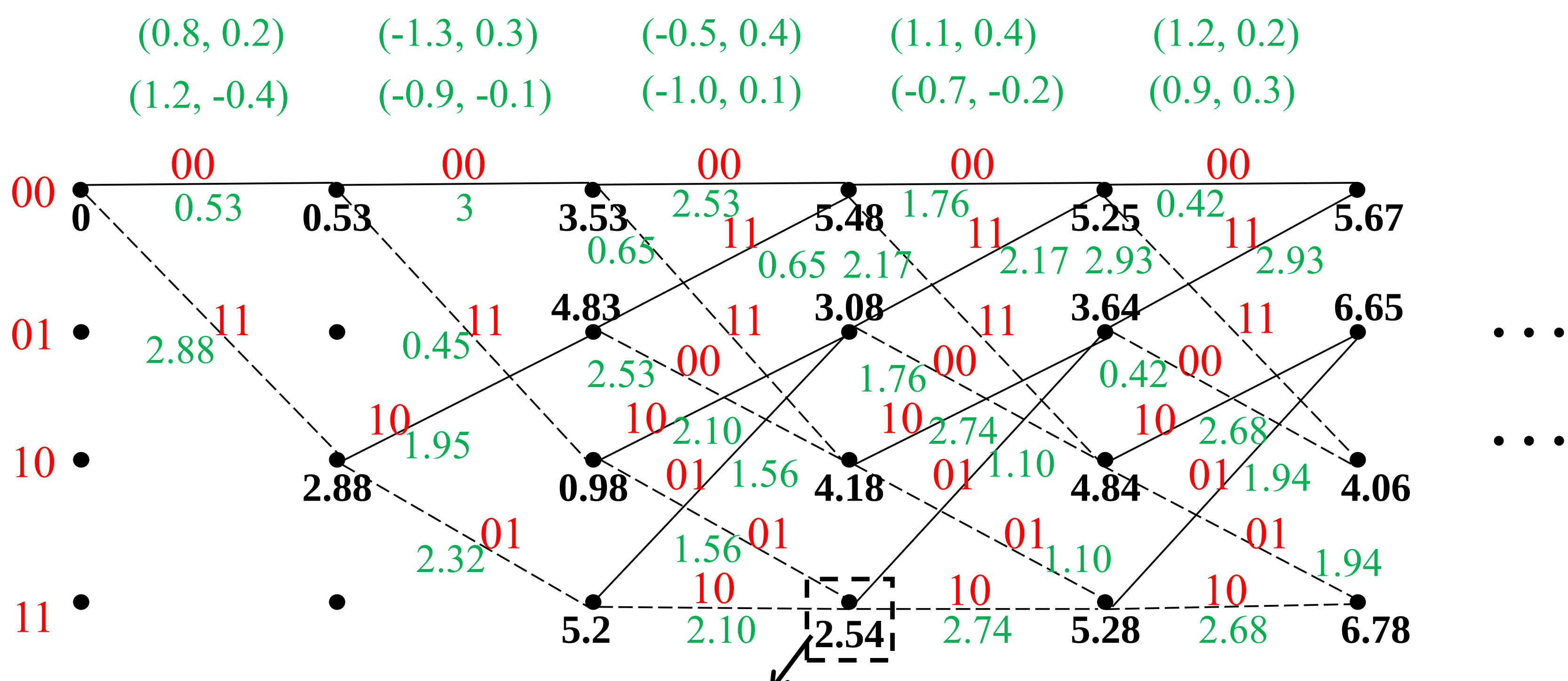
Step 1: Calculate all the branch metrics.

0 → (1, 0)
1 → (-1, 0)



§ 3.3 Viterbi Decoding Algorithm

Step 2: Calculate the path metrics.

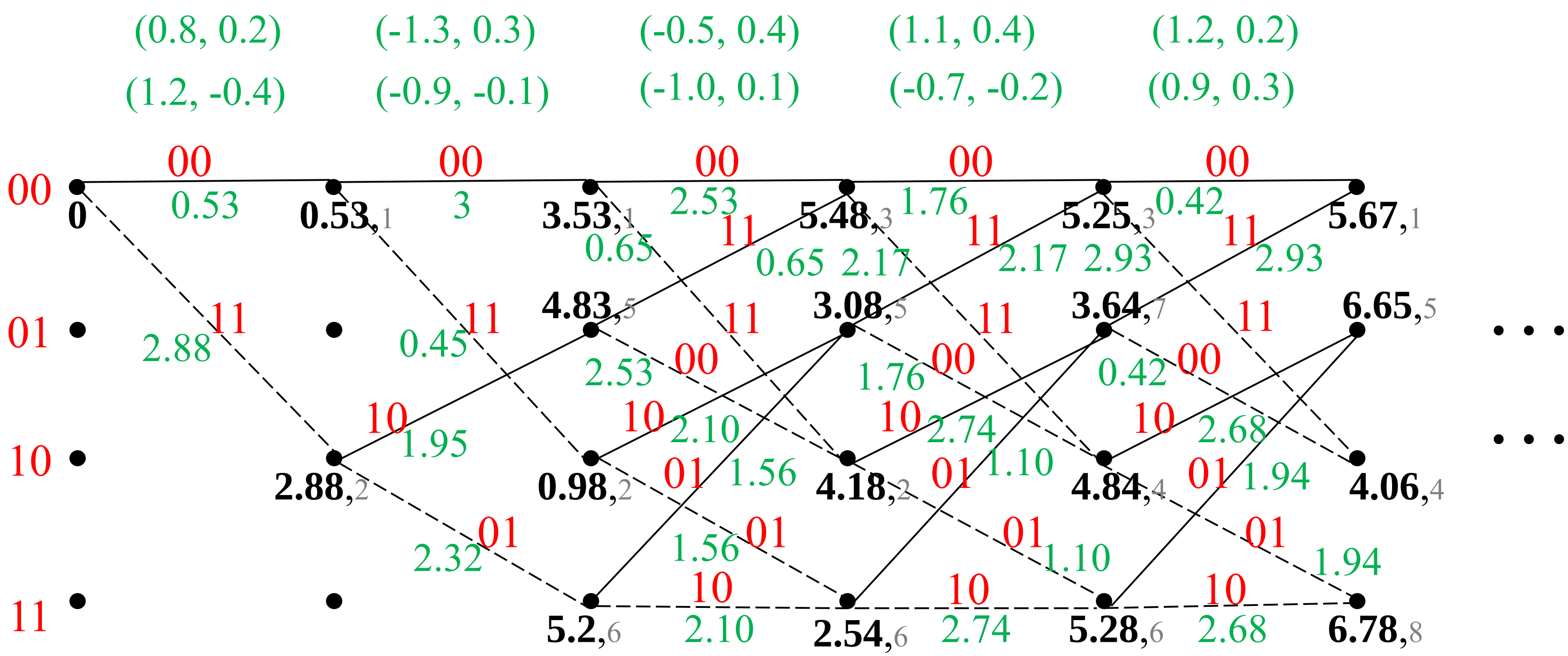


When two paths join in a node, keep the smaller accumulated Euclidean distance.

§ 3.3 Viterbi Decoding Algorithm

Step 3: Pick up the minimal path metric and trace back to determine the message.

Tracing rules: The same as hard-decision Viterbi decoding algorithm.



Decoding output: 0 1 1 0 1

§ 3.3 Viterbi Decoding Algorithm

Branch Metrics Table

0.53	3	2.53	1.76	0.42
∞	∞	0.65	2.17	2.93
2.88	0.45	0.65	2.17	2.93
∞	∞	2.53	1.76	0.42
∞	1.95	2.10	2.74	2.68
∞	∞	1.56	1.10	1.94
∞	2.32	1.56	1.10	1.94
∞	∞	2.10	2.74	2.68

Path Metrics Table

0	0.53	3.53	5.48	5.25	5.67
∞	∞	4.83	3.08	3.64	6.65
∞	2.88	0.98	4.18	4.84	4.06
∞	∞	5.2	2.54	5.28	6.78

Trellis Transition ID Table

1	1	3	3	1
×	5	5	7	5
2	2	2	4	4
×	6	6	6	8

§ 3.3 Viterbi Decoding Algorithm

Free distance of convolutional code

- A convolutional code's performance is determined by its free distance.
- Free distance

$$d_{free} = \min\{d_{Ham}(\bar{C}_1, \bar{C}_2), \bar{C}_1 \neq \bar{C}_2\}$$

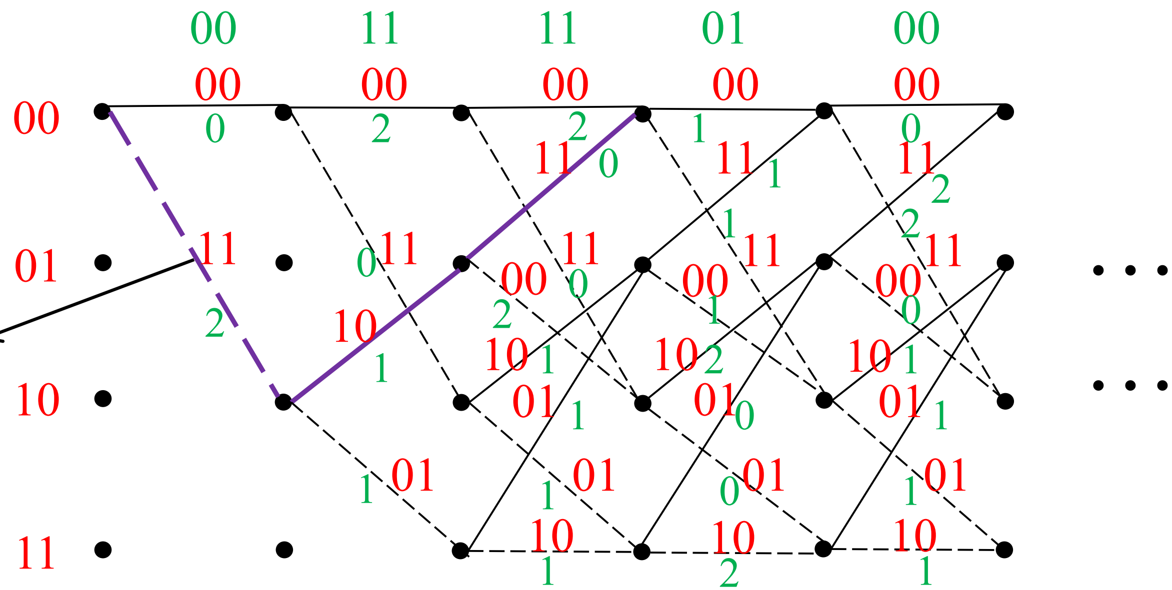
- With knowing $\bar{C}' = [0\ 0\ 0\ \cdots 0]$ is also a convolutional codeword.

$$d_{free} = \min\{weight(\bar{C}), \bar{C} \neq \mathbf{0}\}$$

Hence, it is the minimum weight of all finite length paths in the Viterbi trellis that diverge from and emerge with the all zero state.

§ 3.3 Viterbi Decoding Algorithm

Hence, it is the minimum weight of all finite length paths in the Viterbi trellis that diverge from and emerge with the all zero state.



$d_{free} = 5$

***Remark*:** Convolutional code with a large number of states will have a great d_{free} , and hence stronger error-correction capability.

§ 3.3 Viterbi Decoding Algorithm

***Remark*:** Convolutional code is more competent in correcting spread errors, but not bursty errors.

error

E.g., with $\bar{R}_1 = [0 \quad 1 \quad e \quad 1 \quad \boxed{e} \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad e \quad 1]$

and $\bar{R}_2 = [0 \quad 1 \quad 0 \quad e \quad e \quad e \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1]$,

Viterbi algorithm is more competent in correcting received vector $\bar{R}_1$

§ 3.4 BCJR Decoding Algorithm

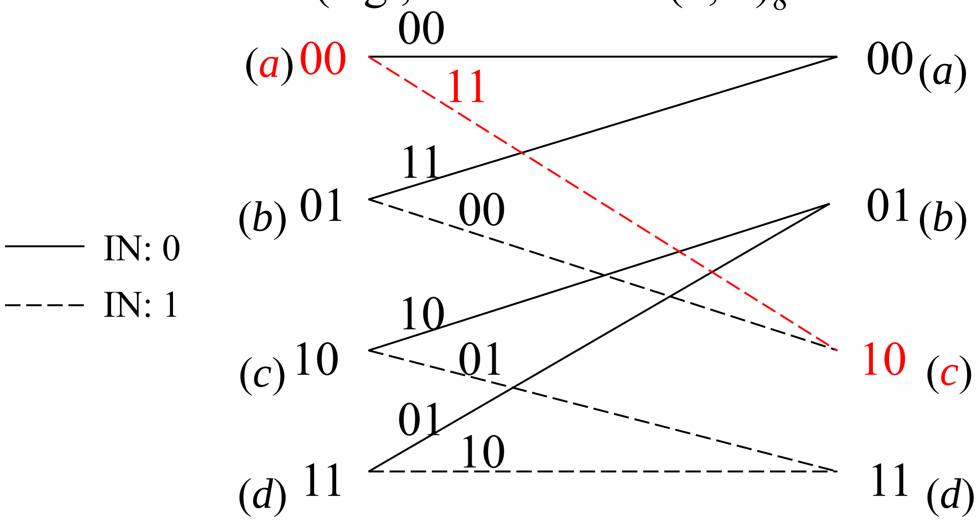
- Hard-decision Viterbi algorithm: a Hard-In-Hard-Out (HIHO) decoding.
  Soft-decision Viterbi algorithm: a Soft-In-Hard-Out (SIHO) decoding.
  BCJR Algorithm: a Soft-In-Soft-Out (SISO) decoding.

- A Soft-In-Soft-Out (SISO) decoding algorithm that takes probabilities as the input and delivers probabilities as the output.

- With an attempt to deliver both the *a posteriori* probabilities of $P(c_t|\bar{y})$ and $P(m_{t'}|\bar{y})$, it is also called the maximum *a posteriori* (MAP) algorithm.
  - c_t — convolutional coded bit, $t = 1, 2, \cdots n$.
  - $m_{t'}$ — information bit, $t' = 1, 2, \cdots k$.
  - $\bar{y}$ — received symbol vector.

§ 3.4 BCJR Decoding Algorithm

- In a trellis (e.g., trellis of the $(7, 5)_8$ conv. code).



The (IN, OUT, current state, next state) tuple happens as an entity.

That says at time instant t'

Σ Prob [trellis transition w.r.t. an input $\boldsymbol{\theta}$] = Prob $[m_{t'} = \boldsymbol{\theta}]$, $\boldsymbol{\theta} \in \{0, 1\}$.

Σ Prob [trellis transition w.r.t. an output of $\boldsymbol{\theta}$] = Prob $[c_{t'}^{1(2)} = \boldsymbol{\theta}]$, $\boldsymbol{\theta} \in \{0, 1\}$.

We seek to determine all the Prob [trellis transition w.r.t. an input of $\boldsymbol{\theta}$] at time t' to know

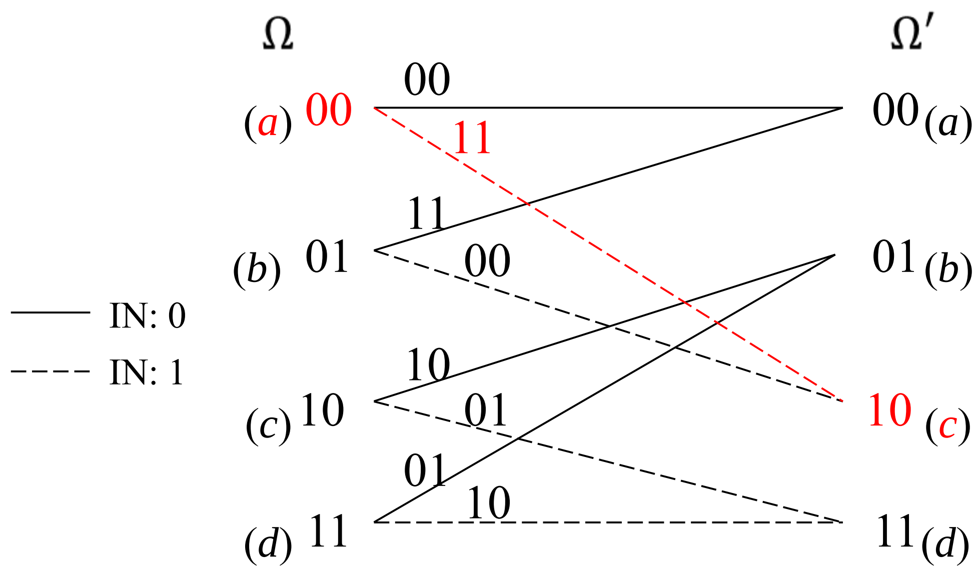
$$P(m_{t'} = \boldsymbol{\theta} \,|\, \bar{y}).$$

We seek to determine all the Prob [trellis transition w.r.t. an output of $\boldsymbol{\theta}$] at time t' to know

$$P(c_{t'}^{1(2)} = \boldsymbol{\theta} \,|\, \bar{y}).$$

§ 3.4 BCJR Decoding Algorithm

A Viterbi trellis snapshot at time instant t':

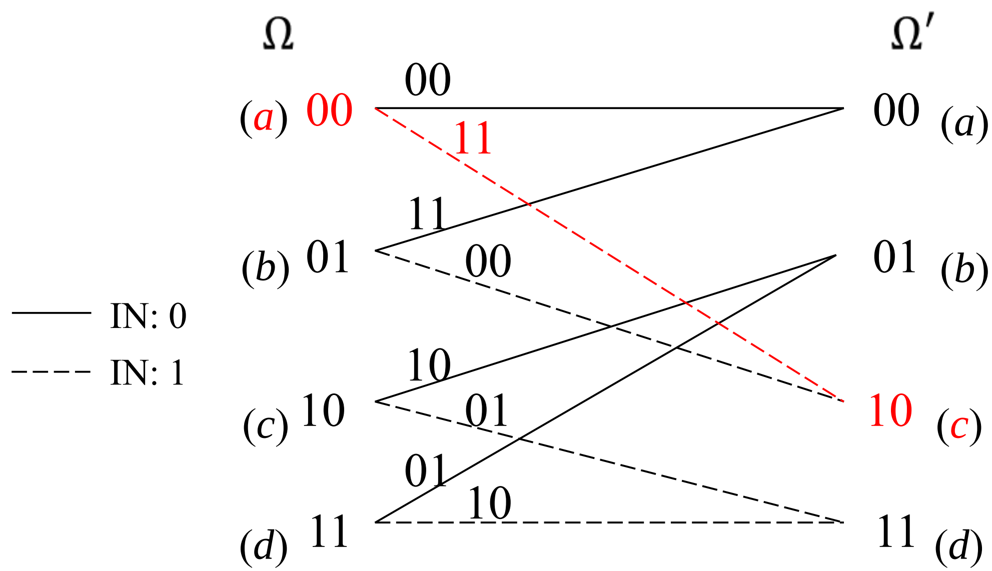


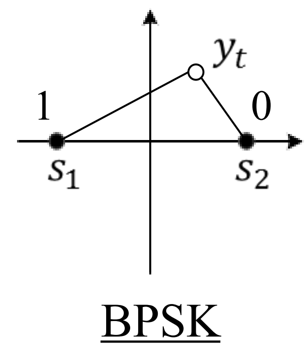
- For a rate half conv. code, $m_{t'} \longrightarrow c_{t'}^1, c_{t'}^2$
- Trellis state transition probability: $(\Omega, \Omega') \in \{a,\ b,\ c,\ d\}$

$$\Gamma_{\Omega \to \Omega'} = P_a(m_{t'}) \cdot P_{ch}(c_{t'}^1) \cdot P_{ch}(c_{t'}^2)$$

§ 3.4 BCJR Decoding Algorithm

- Determine the state transition probabilities.





$$\Gamma_{\Omega\to\Omega'} = P_a(m_{t'}) \cdot P_{ch}(c_{t'}^1) \cdot P_{ch}(c_{t'}^2)$$

A priori prob. of information bit. E.g., w/o knowledge of $m_{t'}$,

$$P_a(m_{t'} = 0) = P_a(m_{t'} = 1) = 0.5.$$

Channel observations:

E.g., BPSK is used for modulation.

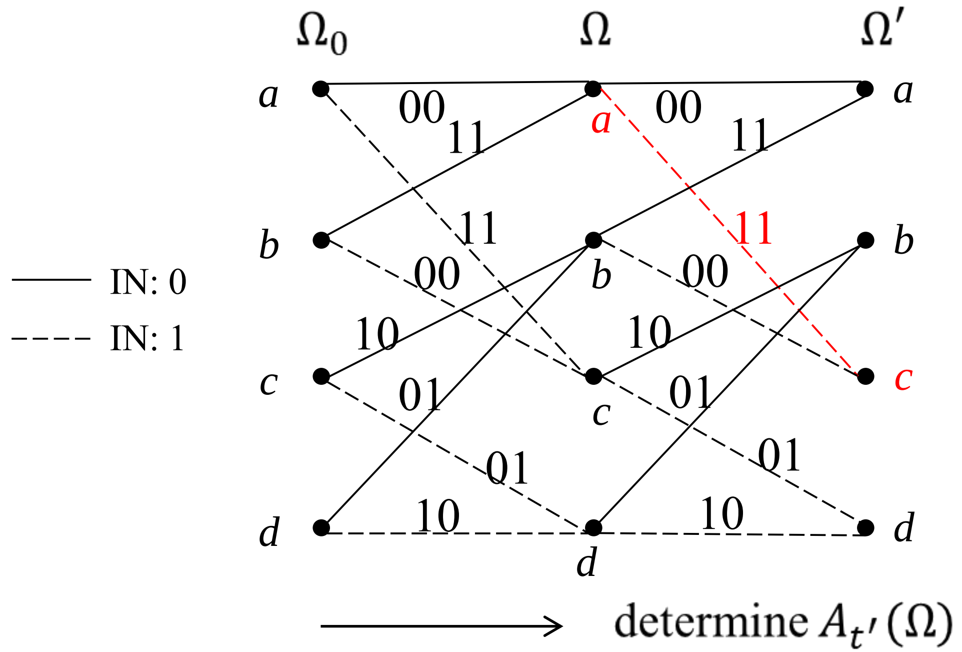
$$P_{ch}(c_{t'}^1 = 0) = P(y_t \mid c_{t'}^1 = 0) = \frac{1}{\pi N_0} \exp\left(-\frac{||y_t - s_2||^2}{N_0}\right)$$

$$P_{ch}(c_{t'}^1 = 1) = P(y_t \mid c_{t'}^1 = 1) = \frac{1}{\pi N_0} \exp\left(-\frac{||y_t - s_1||^2}{N_0}\right)$$

$P_{ch}(c_{t'}^2)$ can be calculated similarly.

§ 3.4 BCJR Decoding Algorithm

- Determine the probability of each beginning state.



- Probability of beginning a trellis transition ($\Omega \rightarrow \Omega'$) from state Ω (Determined by a forward trace).

$$A_{t'}(\Omega) = N_A \sum_{(\Omega_0, \Omega)} A_{t'-1}(\Omega_0) \cdot \Gamma_{\Omega_0 \rightarrow \Omega},$$
$$t' = 1, 2, \cdots, k.$$

N_A: Normalization factor that ensures $A_{t'}(a) + A_{t'}(b) + A_{t'}(c) + A_{t'}(d) = 1$.

- Knowing the Viterbi trellis starts from the all-zero state, we initialize:

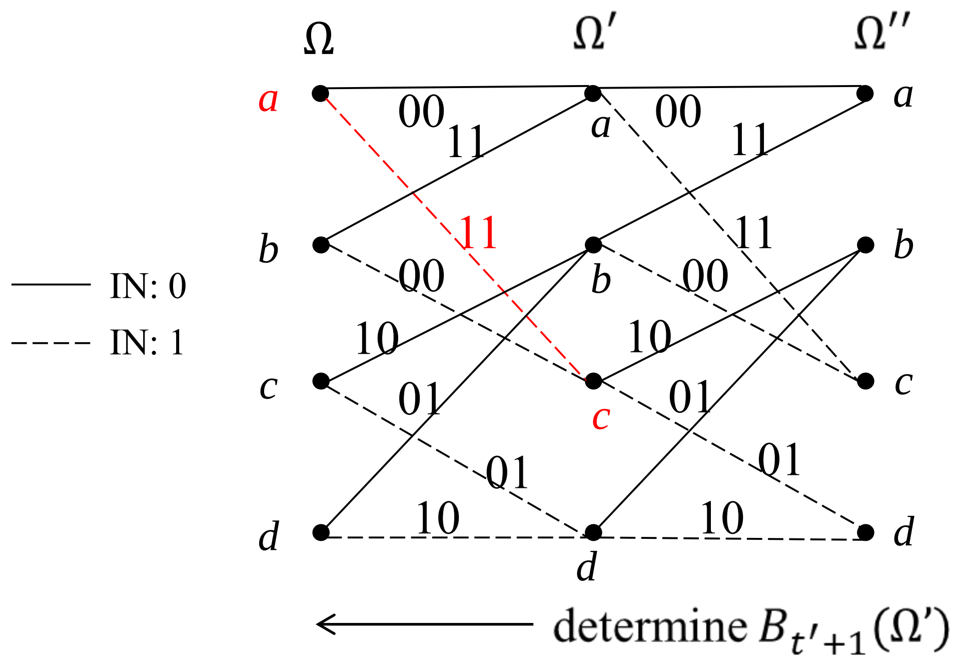
$$A_0(a) = 1, \text{ and } A_0(b) = A_0(c) = A_0(d) = 0.$$

- E.g., in the highlighted trellis transition

$$A_{t'}(a) = A_{t'-1}(a) \cdot \Gamma_{a \rightarrow a} + A_{t'-1}(b) \cdot \Gamma_{b \rightarrow a}.$$

§ 3.4 BCJR Decoding Algorithm

- Determine the probability of each ending state.



- Probability of ending the trellis transition ($\Omega \rightarrow \Omega'$) at state Ω' (Determined by a backward trace).

  $B_{t'+1}(\Omega') = N_B \sum_{(\Omega', \Omega'')} B_{t'+2}(\Omega'') \cdot \Gamma_{\Omega' \rightarrow \Omega''}$.

  N_B: Normalization factor that ensures $B_{t'+1}(a) + B_{t'+1}(b) + B_{t'+1}(c) + B_{t'+1}(d) = 1$.

- By ensuring after encoding, the shift registers (encoder) are restored to the all zero state (achieved by bit tailing), we can initialize:

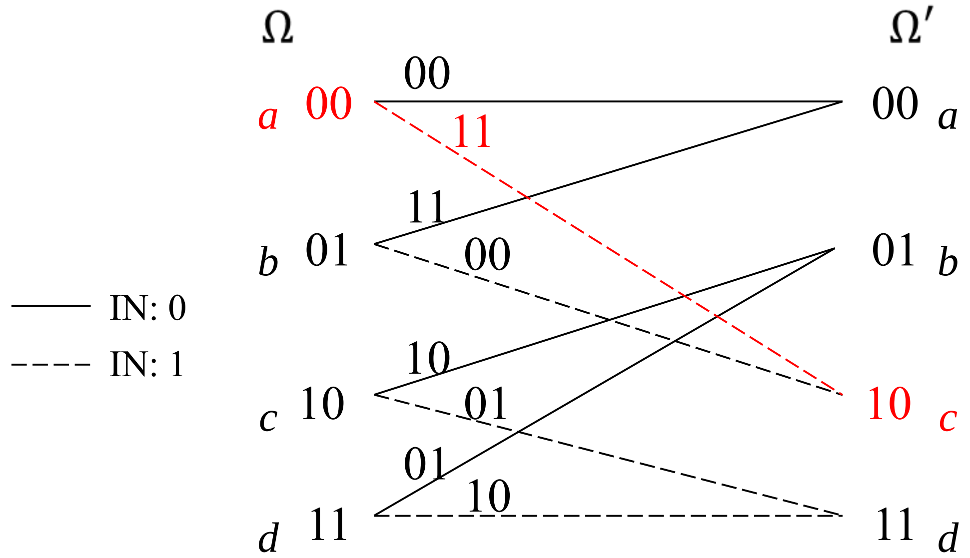
$$B_{k+1}(a) = 1, \text{ and } B_{k+1}(b) = B_{k+1}(c) = B_{k+1}(d) = 0.$$

- E.g., in the highlighted trellis transition

$$B_{t'+1}(c) = B_{t'+2}(b) \cdot \Gamma_{c \rightarrow b} + B_{t'+2}(d) \cdot \Gamma_{c \rightarrow d}.$$

§ 3.4 BCJR Decoding Algorithm

- Determine the *a posteriori* probability of each information bit



- After the ***Forward Trace*** and ***Backward Trace***, we obtain all the $A_{t'}(\Omega)$, $B_{t'+1}(\Omega')$ and $\Gamma_{\Omega\to\Omega'}$ of each time instant t'. We can now determine the *a posteriori* probabilities $P(m_{t'}|\bar{y})$ for each information bit as

$$P(m_{t'}=0|\bar{y}) = N_P \sum_{\underline{(\Omega,\Omega')_0}} A_{t'}(\Omega)\cdot\Gamma_{\Omega\to\Omega'}\cdot B_{t'+1}(\Omega')$$

State transition indicated by ——

$$P(m_{t'}=1|\bar{y}) = N_P \sum_{\underline{(\Omega,\Omega')_1}} A_{t'}(\Omega)\cdot\Gamma_{\Omega\to\Omega'}\cdot B_{t'+1}(\Omega')$$

State transition indicated by ----

N_P: Normalization factor that ensures $P(m_{t'}=0|\bar{y}) + P(m_{t'}=1|\bar{y}) = 1$.

§ 3.4 BCJR Decoding Algorithm

- E.g.,

$$P(m_{t'} = 0|\bar{y}) = N_P \cdot (A_{t'}(a) \cdot \Gamma_{a\to a} \cdot B_{t'+1}(a) + A_{t'}(b) \cdot \Gamma_{b\to a} \cdot B_{t'+1}(a)$$
$$A_{t'}(c) \cdot \Gamma_{c\to b} \cdot B_{t'+1}(b) + A_{t'}(d) \cdot \Gamma_{d\to b} \cdot B_{t'+1}(b)).$$

$$N_P = P(m_{t'} = 0|\bar{y}) + P(m_{t'} = 1|\bar{y}).$$

- Decision based on the *a posteriori* probabilities.

$$\hat{m}_{t'} = 0, \text{ if } P(m_{t'} = 0|\bar{y}) \geq P(m_{t'} = 1|\bar{y})$$

$$\hat{m}_{t'} = 1, \text{ if } P(m_{t'} = 1|\bar{y}) > P(m_{t'} = 0|\bar{y}).$$

§ 3.4 BCJR Decoding Algorithm

Example 3.6. With the same transmitted codeword and received symbols of **Example 3.5**, use the BCJR algorithm to decode it.

With the received symbols, we can determine

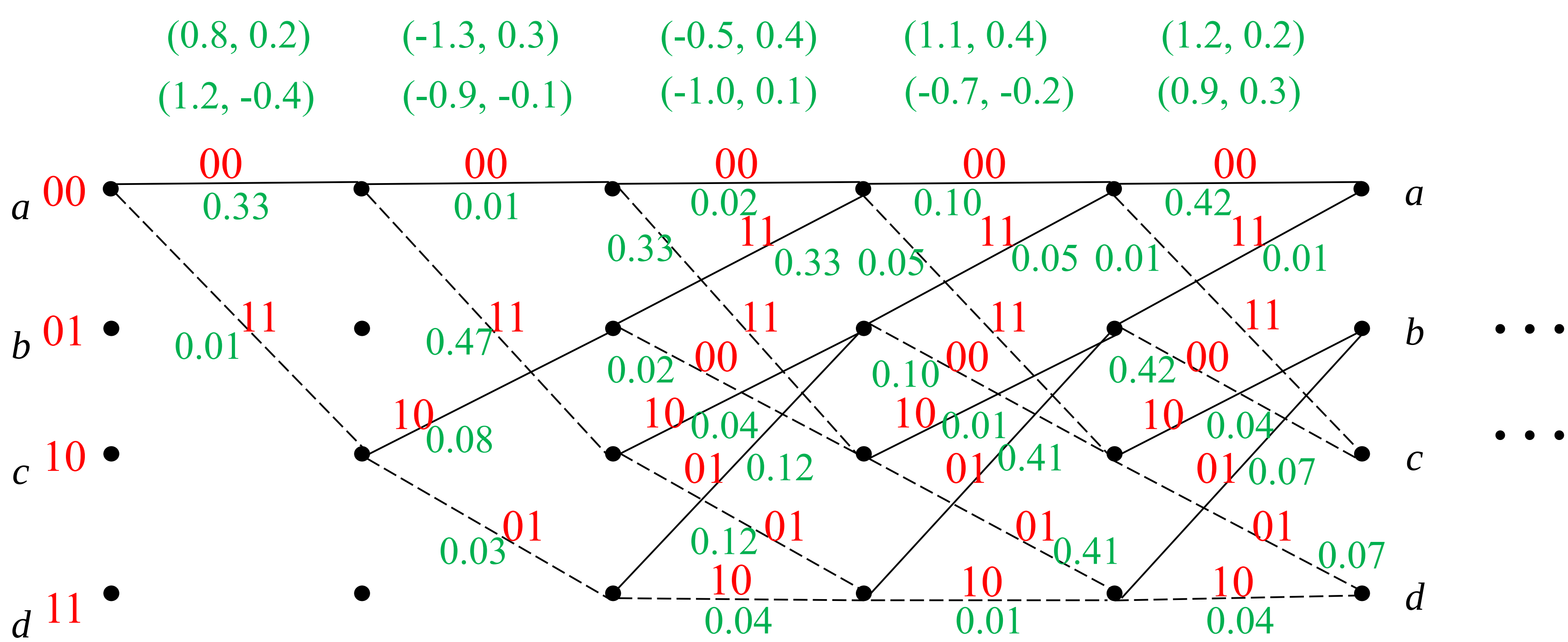
$$\begin{cases} P_{ch}(c_1^1 = 0) = 0.83 \\ P_{ch}(c_1^1 = 1) = 0.17 \end{cases} \quad \begin{cases} P_{ch}(c_1^2 = 0) = 0.92 \\ P_{ch}(c_1^2 = 1) = 0.08 \end{cases} \quad \begin{cases} P_{ch}(c_2^1 = 0) = 0.07 \\ P_{ch}(c_2^1 = 1) = 0.93 \end{cases} \quad \begin{cases} P_{ch}(c_2^2 = 0) = 0.14 \\ P_{ch}(c_2^2 = 1) = 0.86 \end{cases}$$

$$\begin{cases} P_{ch}(c_3^1 = 0) = 0.27 \\ P_{ch}(c_3^1 = 1) = 0.73 \end{cases} \quad \begin{cases} P_{ch}(c_3^2 = 0) = 0.12 \\ P_{ch}(c_3^2 = 1) = 0.88 \end{cases} \quad \begin{cases} P_{ch}(c_4^1 = 0) = 0.90 \\ P_{ch}(c_4^1 = 1) = 0.10 \end{cases} \quad \begin{cases} P_{ch}(c_4^2 = 0) = 0.20 \\ P_{ch}(c_4^2 = 1) = 0.80 \end{cases}$$

$$\begin{cases} P_{ch}\left(c_5^1 = 0\right) = 0.92 \\ P_{ch}\left(c_5^1 = 1\right) = 0.08 \end{cases} \quad \begin{cases} P_{ch}\left(c_5^2 = 0\right) = 0.86 \\ P_{ch}\left(c_5^2 = 1\right) = 0.14 \end{cases}$$

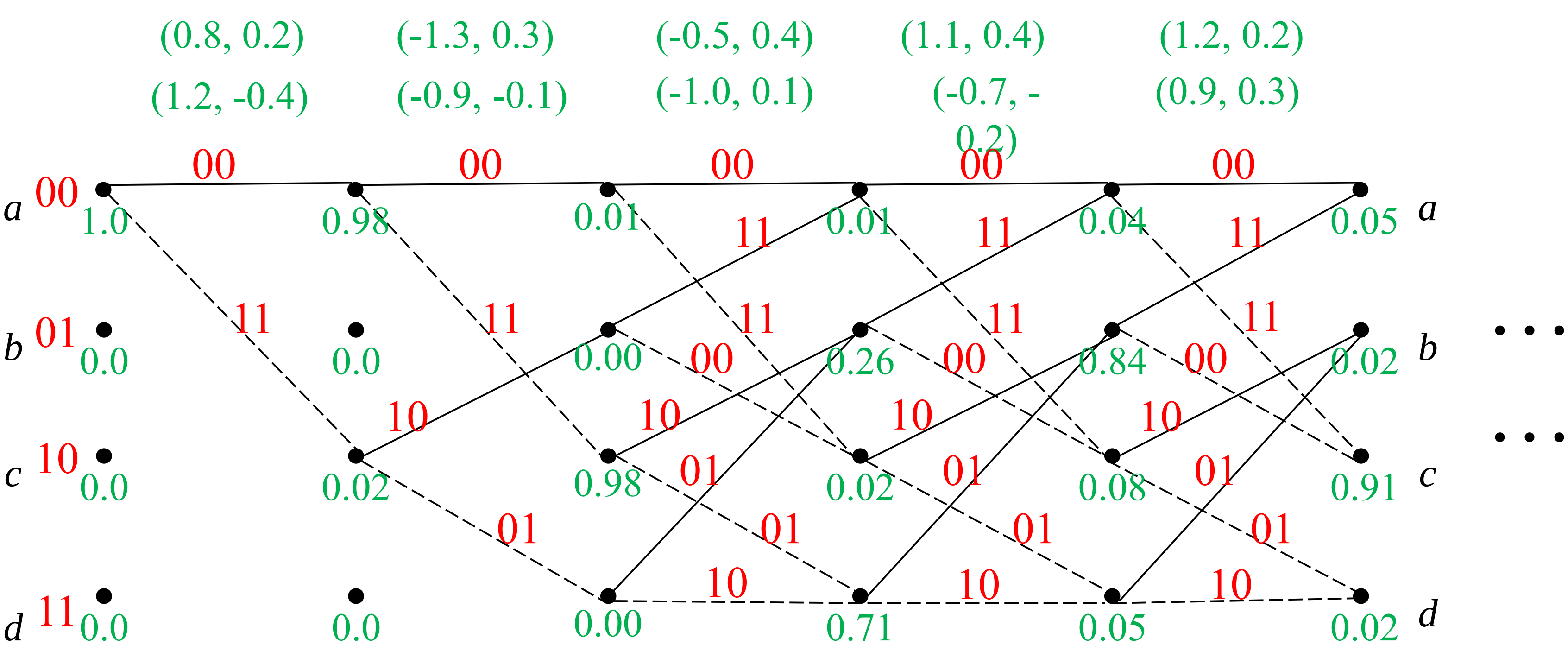
§ 3.4 BCJR Decoding Algorithm

Step 1: Determine $\Gamma_{\Omega\to\Omega'}$ of all transitions.



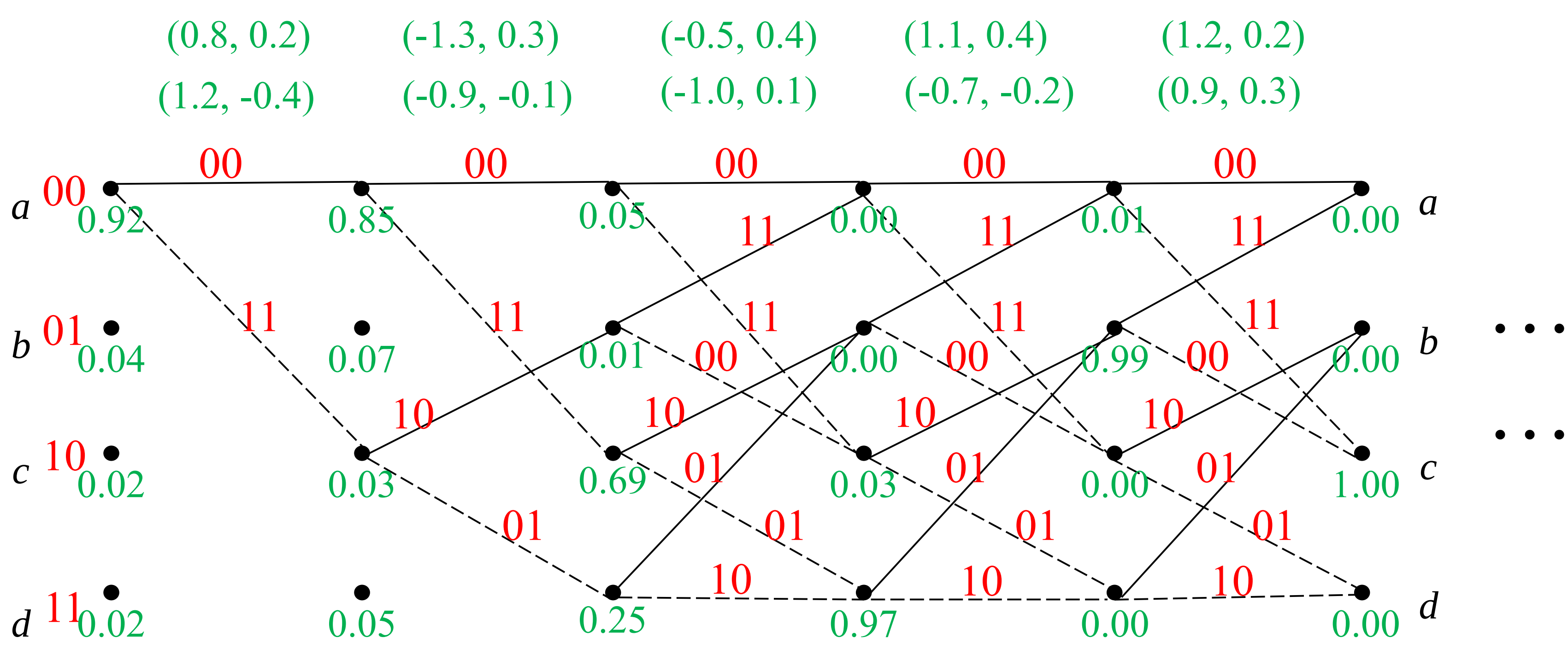
§ 3.4 BCJR Decoding Algorithm

Step 2: Forward trace, determine $A_{t'}(\Omega)$ of values.



§ 3.4 BCJR Decoding Algorithm

Step 3: Backward Trace, determine $B_{t'+1}(\Omega')$ of values.



Assume we know the trellis ends at state c.

§ 3.4 BCJR Decoding Algorithm

Step 4: Determine the *a posteriori* probabilities of each information bit.

$$\begin{cases} P(m_1 = 0|\bar{y}) = 1.00 \\ P(m_1 = 1|\bar{y}) = 0.00 \end{cases} \Longrightarrow \hat{m}_1 = 0$$

$$\begin{cases} P(m_2 = 0|\bar{y}) = 0.00 \\ P(m_2 = 1|\bar{y}) = 1.00 \end{cases} \Longrightarrow \hat{m}_2 = 1$$

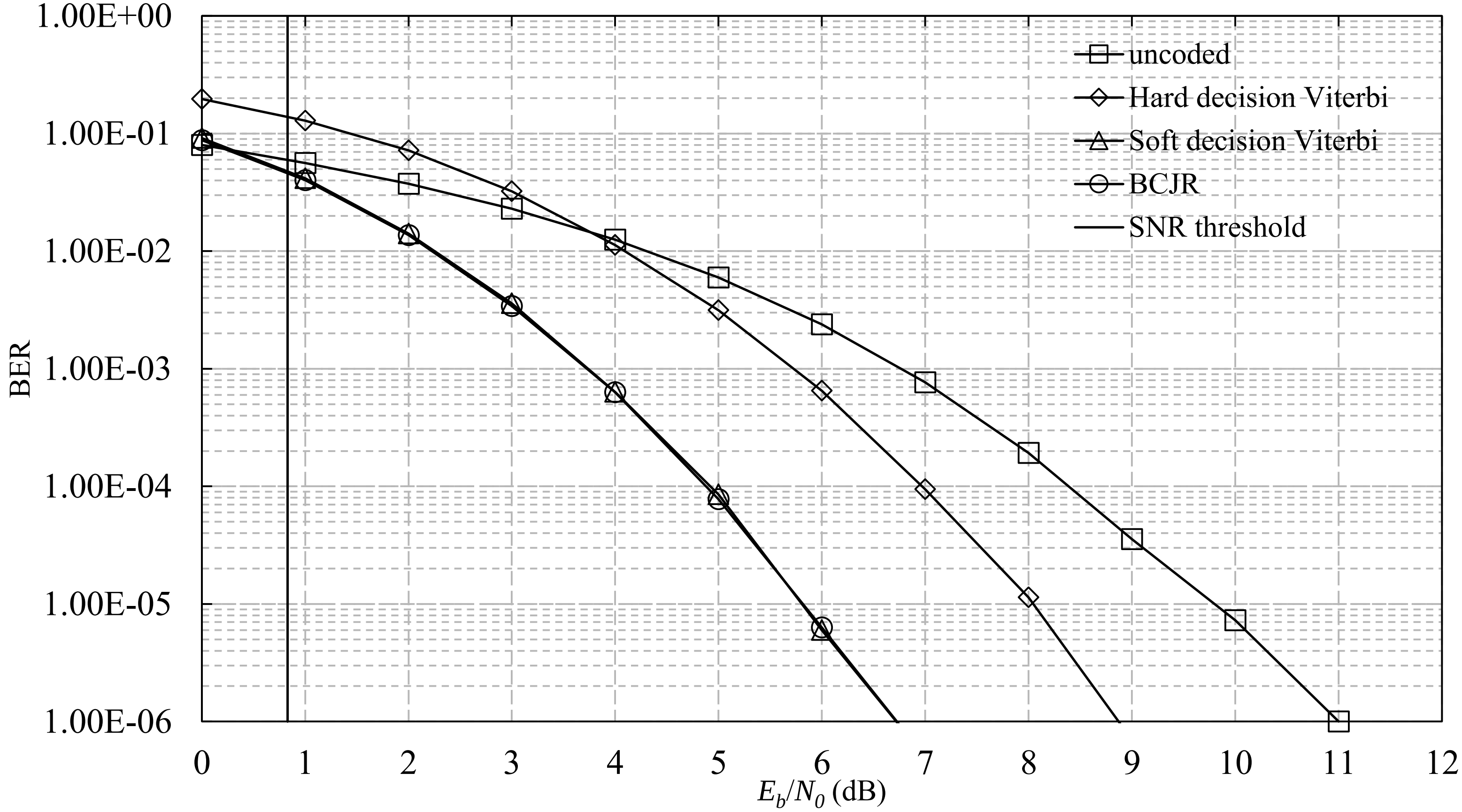
$$\begin{cases} P(m_3 = 0|\bar{y}) = 0.00 \\ P(m_3 = 1|\bar{y}) = 1.00 \end{cases} \Longrightarrow \hat{m}_3 = 1$$

$$\begin{cases} P(m_4 = 0|\bar{y}) = 1.00 \\ P(m_4 = 1|\bar{y}) = 0.00 \end{cases} \Longrightarrow \hat{m}_4 = 0$$

$$\begin{cases} P(m_5 = 0|\bar{y}) = 0.00 \\ P(m_5 = 1|\bar{y}) = 1.00 \end{cases} \Longrightarrow \hat{m}_5 = 1$$

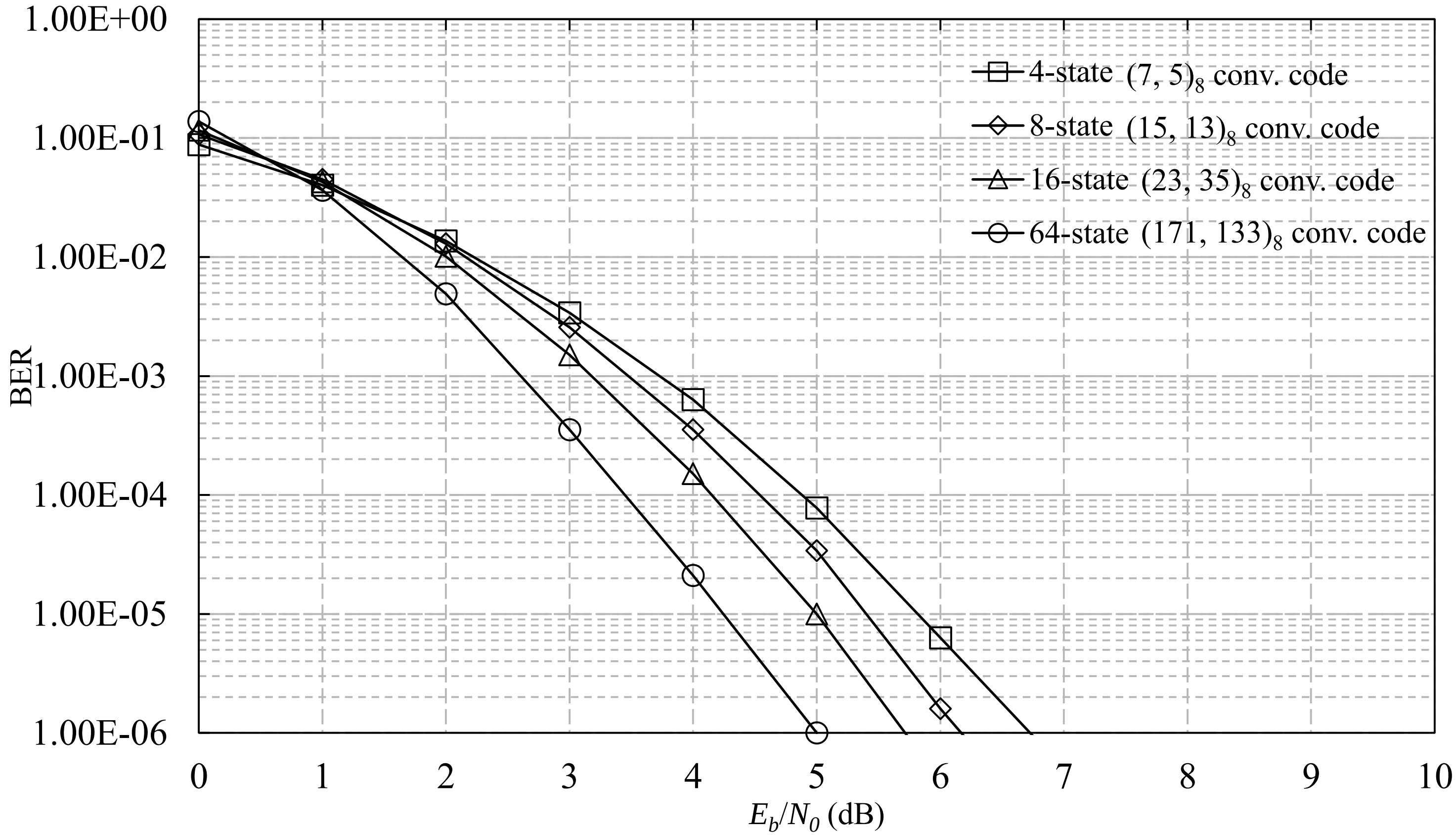
§ 3.4 BCJR Decoding Algorithm

BER performance of $(7, 5)_8$ conv. code over AWGN channel using BPSK.



§ 3.4 BCJR Decoding Algorithm

BER performance of different conv. code over AWGN channel using BPSK.



§ 3.5 Trellis Coded Modulation

- Convolutional code enables reliable communications. But as a channel code, its error-correction function is on the expense of spectral efficiency.
- Spectral efficiency $(\eta) = \frac{Nr.\,of\ information\ bits}{transmitted\ symbol}$
- E.g., an uncoded system using BPSK

  η = 1 info bits/symbol

  A rate 1/2 conv. coded system using BPSK

  η = 0.5 info bits/symbol

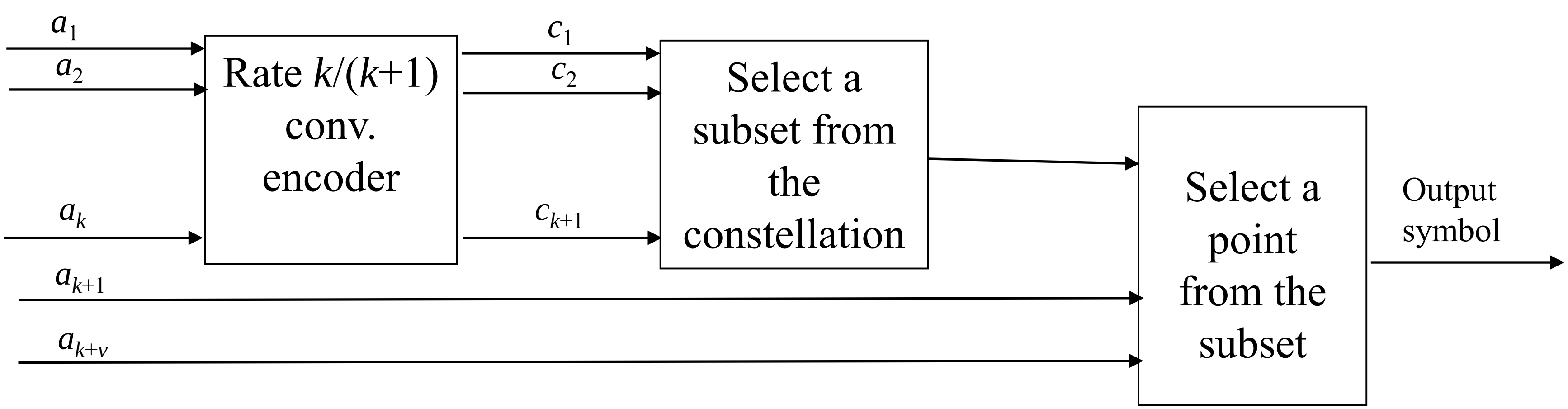
- Can we achieve reliable and yet spectrally efficient communication?

  Solution: Trellis Coded Modulation (TCM) that integrates a conv. code with a high order modulation [3].

[3] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 55-67, 1982.

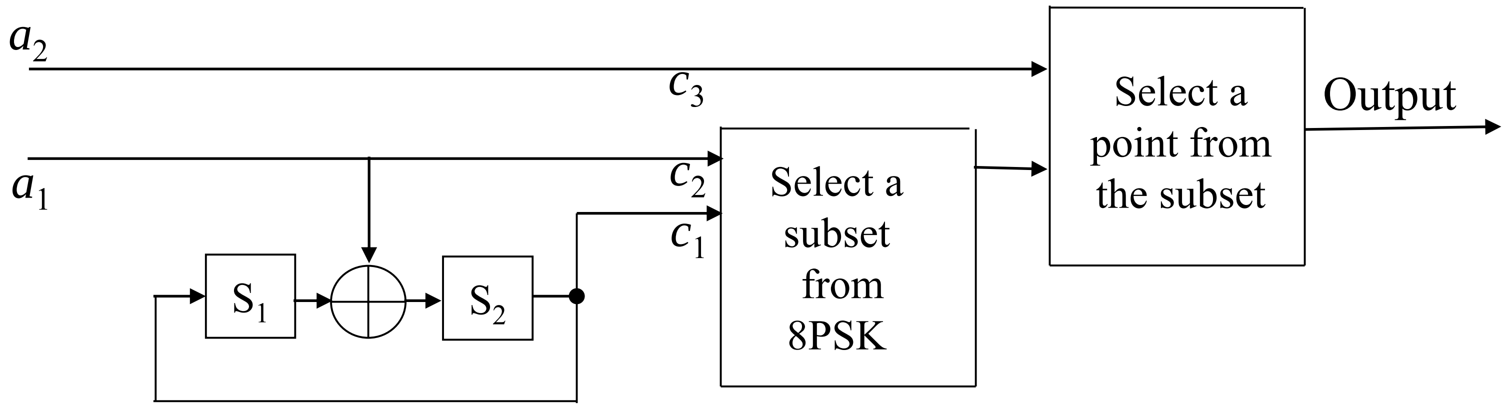
§ 3.5 Trellis Coded Modulation

- A general structure of the TCM scheme



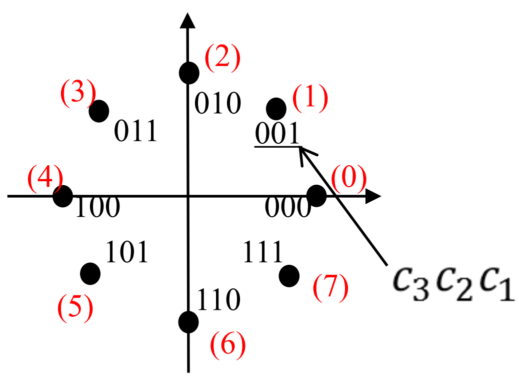
§ 3.5 Trellis Coded Modulation

- A rate 2/3 TCM code.



Rate ½ 4-state Convolutional Code

8PSK Constellation



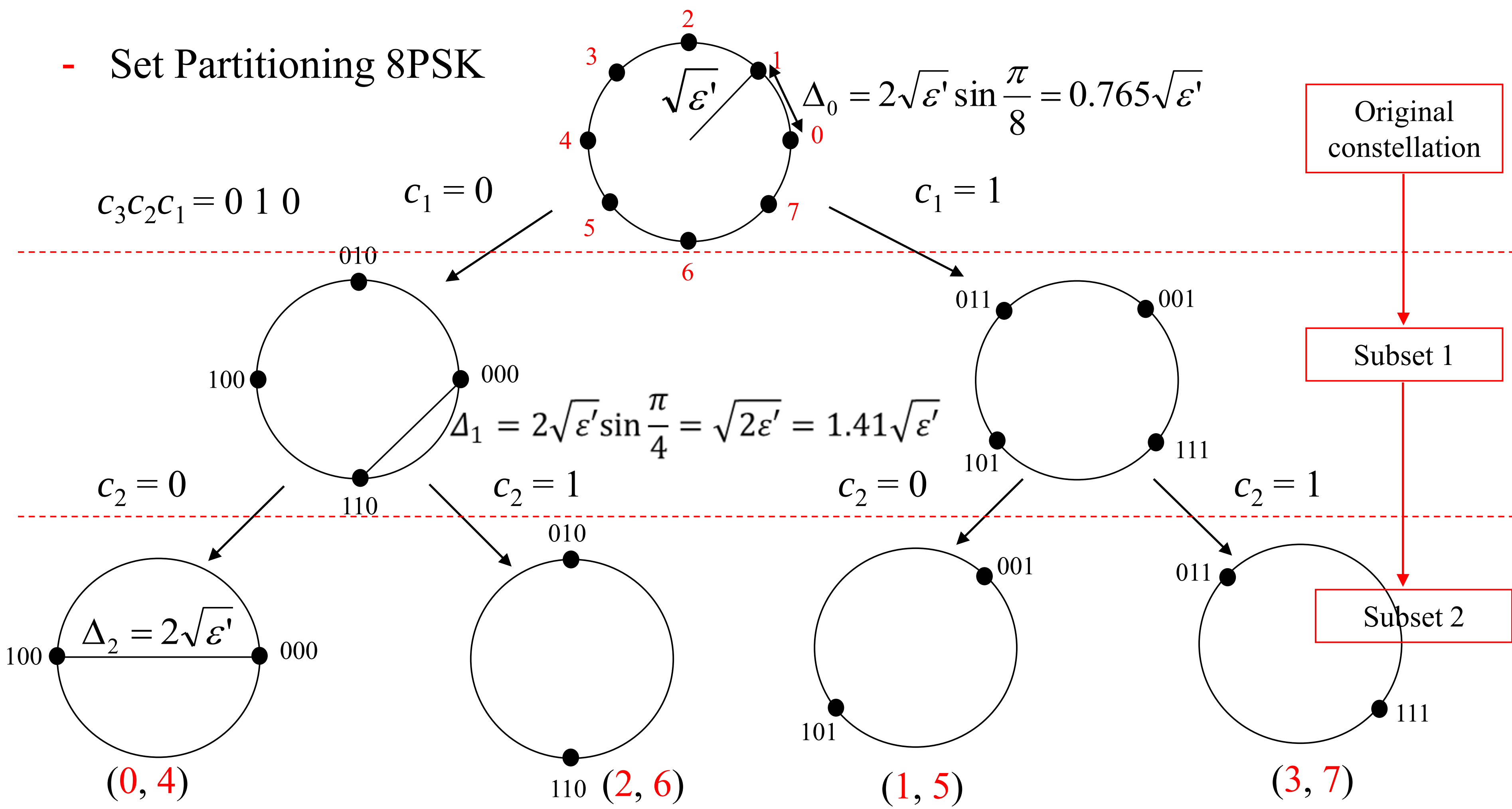
§ 3.5 Trellis Coded Modulation

- State table of the rate 2/3 TCM code

Input		Current State		Next State		Output			Symbol
a_1	a_2	S_1	S_2	S_1'	S_2'	c_1	c_2	c_3	**8PSK sym**
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	0	2
0	1	0	0	0	0	0	0	1	4
1	1	0	0	0	1	0	1	1	6
0	0	0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	1	0	3
0	1	0	1	1	0	1	0	1	5
1	1	0	1	1	1	1	1	1	7
0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	1	0	2
0	1	1	0	0	1	0	0	1	4
1	1	1	0	0	0	0	1	1	6
0	0	1	1	1	1	1	0	0	1
1	0	1	1	1	0	1	1	0	3
0	1	1	1	1	1	1	0	1	5
1	1	1	1	1	0	1	1	1	7

§ 3.5 Trellis Coded Modulation

- Set Partitioning 8PSK



§ 3.5 Trellis Coded Modulation

- Set Partitioning 8PSK

By doing set partitioning, the minimum distance between point within a subset is increasing as: $\Delta_0 < \Delta_1 < \Delta_2$.

Set Partitioning

Original constellation

$$\Delta_0 = d(0,1) = 2\sqrt{\varepsilon'}\sin\frac{\pi}{8} = 0.765\sqrt{\varepsilon'}$$

Subset 1

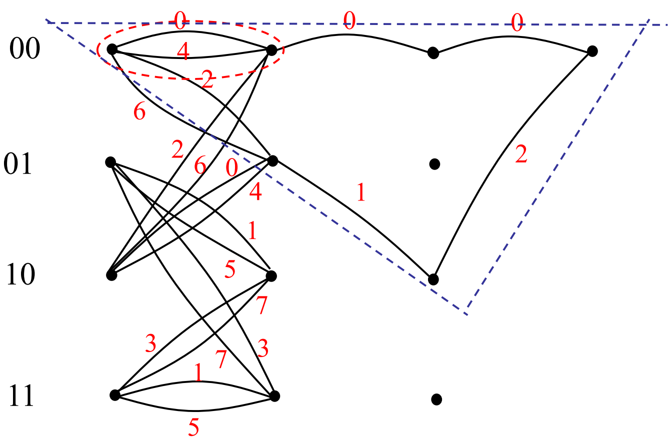
$$\Delta_1 = d(0,2) = \sqrt{2\varepsilon'} = 1.414\sqrt{\varepsilon'}$$

Subset 2

$$\Delta_2 = d(0,4) = 2\sqrt{\varepsilon'}$$

§ 3.5 Trellis Coded Modulation

- Viterbi trellis of the rate 2/3 TCM code



For diverse/remerge transition:

$$d_{free}^2 = [d^2(0,2) + d^2(0,1) + d^2(0,2)]$$
$$= 2\varepsilon' + (2 - \sqrt{2})\varepsilon' + 2\varepsilon' = 4.586\varepsilon'$$

For parallel transition:

$$d_{free}^2 = d^2(0,4) = 4\varepsilon'$$

Choose the smaller one as the free distance of the code:

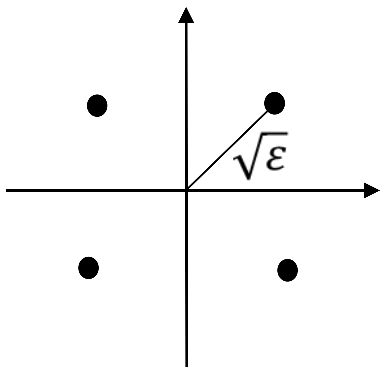
$$d_{free}^2 = 4\varepsilon'$$

Remark: Bit $c_3 = 0$ and $c_3 = 1$ result in two parallel transition branches. By doing set partitioning, we are trying to maximize the Euclidean distance between the two parallel branches. So that the free distance of the TCM code can be maximized.

§ 3.5 Trellis Coded Modulation

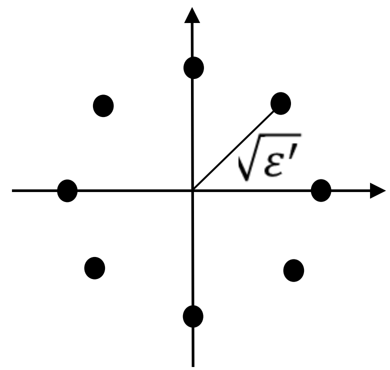
- Asymptotic coding gain over an uncoded system.
- Spectral efficiency (η) = 2 info bits/sym.

uncoded QPSK



$$d_{min}^2 = 2\varepsilon$$

rate 2/3 coded 8PSK



$$d_{free}^2 = 4\varepsilon'$$

Asymptotic coding gain $\gamma = \frac{(d_{free}^2/\varepsilon')}{(d_{min}^2/\varepsilon)} = 2.$

Asymptotic coding gain in dB = $10 \log_{10} \gamma$ = 3 dB.

Remark: With the same transmission spectral efficiency of 2 info bits/sym, the TCM coded system achieves 3 dB coding gain over the uncoded system asymptotically.